AEC Computing and Applied Mathematics Center

AEC RESEARCH AND DEVELOPMENT REPORT

TID-4500
15th Ed.

NYO-9083
PHYSICS

1. FLOW DIAGRAMS
2. THE ESTIMATION OF SIGNIFICANCE

by

R. D. Richtmyer

April 25, 1960

Institute of Mathematical Sciences

NEW YORK UNIVERSITY
NEW YORK, NEW YORK

AEC Computing and Applied Mathematics Center
Institute of Mathematical Sciences
New York University

TID-4500
15th Ed.

NYO-9083
PHYSICS

1. FLOW DIAGRAMS
2. THE ESTIMATION OF SIGNIFICANCE

by

R. D. Richtmyer

April 25, 1960

Contract No. AT(30-1)-1480

NYO-9083
PHYSICS AND
MATHEMATICS

1. FLOW DIAGRAMS[1]

2. THE ESTIMATION OF SIGNIFICANCE

ABSTRACT

Two distinct subjects are discussed in the report.

1. The notion of a mathematical flow diagram for describing a computational procedure, introduced by von Neumann and co-workers in 1946, is discussed. The rules for constructing and interpreting such a diagram are given.

2. Various methods for the estimation of significance in numerical calculations are summarized, with comments.

---

[1] This material was prepared in collaboration with N. C. Metropolis, for possible inclusion as an appendix to a proposed book on Monte Carlo methods.

TABLE OF CONTENTS

1. FLOW DIAGRAMS

2. THE ESTIMATION OF SIGNIFICANCE

1. Introduction.

About 15 years ago, von Neumann and his co-workers felt the need of a more precise way of formulating large computational problems than via the existing mathematical terminology, and they devised a way of representing each problem by a "flow diagram." Since then, the original objective appears to have been lost sight of, to some extent, partly because von Neumann never recorded his ideas on the subject with the completeness and clarity that characterized most of his work.

We feel that it will be useful to set forth once more the original ideas, following quite closely the reports of Burks, Goldstine and von Neumann (1946, 1947), but with minor modifications and amplifications aimed at greater clarity, in line, we hope, with von Neumann's unwritten as well as his written comments on the subject.

The objective was to furnish a precise formulation of an abstract computational procedure,[2] not a mnemonic guide to the operation of a computer. Conceivably, the technique could be useful in other branches of mathematics, where a complicated

---

[2] During the formulation, it is often wise to keep in mind the characteristics (memory size, etc.) of the computer to be used, but it was von Neumann's feeling that a complete abstract formulation should be made, independent of detailed planning for the computer, human or electronic.

pattern of operations or procedures has to be defined.

The need arises because most mathematics is primarily concerned (at least today) with statements of fact, rather than the performance, or even formulation, of specific calculations, numerical or analytic.[3] Mathematics has probably never been concerned with calculations of the size and complexity of many that are nowadays entrusted to computers. One should not expect that standard notations and modes of expression would be suitable or even adequate for that purpose, although they can often be made so by supplementing the formulas with verbal comments such as, "for each j in succession ($j = 0,1,\ldots$), one solves equation (17) for x and substitutes the result into..."; in very large problems, this is cumbersome.

We shall limit discussion to flow diagrams satisfying the following criteria: (1) the intention is to give a complete and precise description of all calculational steps; (2) references to a particular computer, if any, play a secondary role. The first criterion demands, obviously, that all notational conventions used be exactly defined, and we shall define a set of conventions very similar to the ones used originally by Burks, Goldstine, and von Neumann.

A flow diagram is no guarantee that a perfect computer code

---

[3] An important exception would seem to be the tendency, among many mathematicians, to be unhappy about an existence proof that is not constructive - i.e., does not indicate, in principle, by what steps one can construct the particular mathematical object whose existence is being proved.

will be written. Neither is it necessary: some people can plan and code a problem effectively without one; however, many people find it convenient to formulate a problem directly in the form of a flow diagram. (For this purpose, a special pencil is used, which is half pencil and half eraser.) In any case, it is our point of view that only by means of a proper flow diagram can the calculational procedure be adequately described or communicated to another person short of giving a complete computer code or the equivalent.

Diagrams of another kind, called flow charts are often used in addition; they make little attempt to describe the contents of the operation boxes, except by words such as "calculate stream function" or "read new block of data from tape 5", but emphasize mainly the overall sequencing and various circumstances having to do with the operation of a particular computer. They are helpful in checking out and running a problem, but should not be confused with flow diagrams in the original sense.

2. Rules for Flow Diagrams.

The routine of a calculation of sufficient magnitude and complexity to warrant the use of a modern computer usually has a complicated set of interconnections among its various parts, most of which are used many times over in varying contexts. The proper sequencing of the operations in the computer is effected by jump instructions, conditional and unconditional, which provide the interconnections referred to. Each elementary part of the calculation consists of a prearranged sequence of

steps without repetitions, gaps, or alternatives; such a sequence can be specified by a list (for example, in computer code) of the steps, written one over the other in a vertical column, or horizontally one after the other, or in any other linear arrangement. On the other hand, if the description of the complete routine is written in a linear or one-dimensional fashion, or list, the sequence of items in the list fails to represent the sequence of steps in the calculation whenever a jump instruction occurs. In order that the geometrical arrangement of the items should give a topologically[4] true representation of the calculation, two or more dimensions are needed, to display loops, branching, merging, and the like. Two dimensions will always suffice, at least if one allows flow lines to cross each other without interference, as in electrical circuit diagrams. Hence, a two-dimensional figure is natural for a description of the sequencing of a calculation.

The linear portions of the calculation are specified in operation boxes, by means of a symbolism to be described presently, and the boxes are connected by flow lines (bearing arrowheads to indicate their direction), according to a pattern determined by the structure of the problem. Branching is achieved by means of condition boxes, which have two (or more) exits; the computer follows the one or the other exit flow line according to whether some specified condition is satisfied

---

[4] We use this word loosely, and trust that the reader will understand what is meant.

(e.g., according to whether some quantity it has just calculated is positive or negative). The connections are subject to a few rules that are rather self-evident: an operation box has just one exit (except for a box containing stop as its last item, which has no exit); flow lines may merge, but not branch out, between boxes; if two flow lines enter a box, the effect is the same as if they had merged before entering; the items within a box are invariably taken in the order in which they were written (i.e. from top to bottom and, within each row of writing containing two or more items, from left to right within the row), regardless of where the flow lines enter and leave the box. In the case of a special device called a remote connection, to be described later, computer control jumps from one specified point in the diagram to another without following any visible path; this is admittedly contrary to the basic notion of a flow diagram but is necessary, if the diagram will not fit onto a single sheet of paper, and is sometimes convenient for other reasons.

To illustrate the need for careful definition, let us suppose that, in describing a problem, one has written:

$$g = ax^3 + bx^2 + cx + d. \tag{1}$$

This is presumably intended to mean that the numerical values assigned to the variables a, b, c, d, and x, are to be combined in the manner indicated on the right of the equation, and the resulting numerical value is to be assigned to the

variable g. However, in a typical computer calculation, the number of numbers (numerical values) occurring vastly exceeds the number of variables.[5] It is therefore highly likely that all six of the variables a, b, c, d, x, and g, have had numerical values assigned to them at the time equation (1) is encountered; generally these values will not satisfy equation (1), and one of them has to be changed. We have taken it for granted, in the present case, (from our knowledge of the psychology of mathematicians) that it is g, rather than a, b, c, d, or x, that has to change. But, without some clear convention, one is not justified in placing any particular interpretation on (1) as an operational instruction. Even our knowledge of psychology would be of no use if the equation were

(2) $$f(x) = g(y)$$

or

(3) $$a = b$$

(Such equations have been seen in actual problem descriptions!)

One must either redefine the symbol =, for use in flow diagrams, or introduce a new one. Because of the usual static reflexive meaning of the sign =, we prefer[6] to introduce a

---

[5] By a variable is meant a symbol (usually literal) to which different values can be assigned at different times. Owing to the small number of available symbols, most of them generally have to be used many times over.

[6] Many people continue to use the sign = with a definite understanding as to which side of the "equation" is to be changed. This is perfectly sound, but we want to emphasize here that we are dealing with an instruction, not a statement of fact.

new symbol $\rightarrow$ (pronounced "for"), whose meaning is governed by a set of rules, as follows: (the symbol = will reappear later in a different connection).

Rule 1. When the symbol $\rightarrow$ stands between two variables, it means that henceforth, until further notice, the variable on its right is to have assigned to it the numerical value obtained from the variable (or combination of variables) on its left. In other words, the variable to be changed always stands on the right. If the variable on the right appears in the combination of variables on the left, as in the familiar case "$n + 1 \rightarrow n$", the old value is of course to be used there for computing the new one; otherwise, all variables appearing on the left continue to retain their previous values.

With this rule, equation (1) would be rewritten as

$$ax^3 + bx^2 + cx + d \rightarrow g; \tag{4}$$

and equation (3) would be rewritten either as

$$a \rightarrow b,$$

or

$$b \rightarrow a,$$

depending on which is intended.

(We are concerned mainly with abstract flow diagrams. But, if one has a particular computer in mind and wishes to indicate which memory location a quantity is to be stored in, the word "to" is used, not an arrow. This may be combined

with a step of the calculation, if desired; for example, the expression "$x + y \rightarrow z$ to 311" means that the sum of $x + y$ is to be stored in memory location 311 and is to be referred to by the symbol z in subsequent parts of the flow diagram.)

The variable may be a single letter or a compound symbol, like y_n or p_{jk} or $f(x)$, in which a functional relation is implied; we then call it a compound variable. When its arguments (i.e. n, or j and k, or x) have had values assigned to them, it behaves just like a simple variable until the value of one or more of its arguments changes, in which case the following rule applies:

Rule 2. When the value of a variable is modified, this modification is understood to take place automatically in all compound variables that depend on it. For example, if, at some stage in a calculation, n has the value 3, T_n stands for T_3 and T_{n+1} stands for T_4; if a substitution $n + 1 \rightarrow n$ is then made so that n assumes the value 4, the symbol T_n automatically comes to stand for T_4 and T_{n+1} for T_5; it is unnecessary, and in fact incorrect, to write "$T_{n+1} \rightarrow T_n$" after writing "$n + 1 \rightarrow n$", unless one intends not only to alter n but also to alter T_4 so as to make it equal to T_5.

When a compound variable stands on the right of the symbol $\rightarrow$, a further rule is needed to make the meaning unique.

Rule 3. When a compound variable representing a function of one or more arguments stands on the right of the symbol $\rightarrow$,

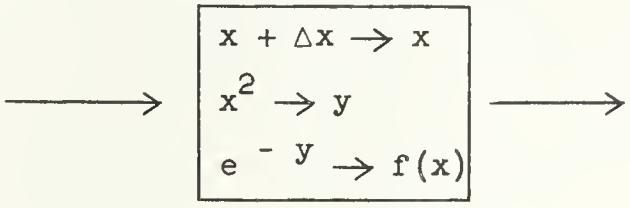
it is the functional value that is to be changed, not an argument.

For example, at a time when j has the value 17 and N the value 6, the expression $N^2 \rightarrow n_j$ means that the variable n_{17} is to be assigned the value 36, even though it might be possible, by changing the value of j, to find some other variable n_j (say n_{13}) that is already equal to 36. Similarly, at a time when x has the value 1.7, the substitution $0.25 \rightarrow f(x)$ means that henceforth $f(1.7) = 0.25$; i.e. it means that the function is to be changed (at this one value of x), not that we should solve the equation $f(x) = 0.25$ to find a new value of x, using the old function.

Occasional exceptions are allowed in the case of functions whose values and notation are permanently established by rigid mathematical convention, so that the argument must change. If $f(x)$ in the last example is the function x^2, the substitution $0.25 \rightarrow x^2$ must clearly mean that the value of x is to be changed from 1.7 to 0.5 (or possibly to -0.5!), and does not imply that we are trying to change the meaning of the function denoted by x^2 at the argument $x = 1.7$, so as to make "$(1.7)^2$" mean "0.25" rather than "2.89." Similarly, the expression $\frac{a}{a^2+1} \rightarrow \sin\theta$ would be taken to require a modification of the value of θ. However, it is best to avoid all exceptions to Rule 3 where there can be possible doubt about the intention.

It was noted above that the items in a box are always to be taken in the order written. This permits one step in a box

to modify variables appearing in later steps in the same box, as for example in



We have been intentionally vague as to how complicated an expression is permitted to occur on the left of the symbol $\to$. We do not wish to be restricted to expression that can be evaluated by an unbroken sequence of coding, because computers vary greatly as to what can be accomplished by a single instruction or elementary operation. Some computers have built-in circuits for square roots and exponentials; whereas, at the other extreme, even division requires a coded subroutine in some. We are aiming at an abstract description of a calculation, independent of computers, and the only requirement in the present connection is that, if there is any doubt as to what calculational steps are needed to evaluate an expression used, the diagram should be expanded to show the steps in more detail.

If we regard an instantaneous state of a calculation as being determined, for purposes of abstract description, when some or all of the variables have specified numerical values assigned to them, it is clear that the calculation progresses from one state to another only during a step described by an expression containing the sign $\to$. We call such a step a (generalized) substitution, because, generally, one numerical value is

substituted for another as the meaning of some variable.

It may be noted parenthetically that the instantaneous state of a given computer may have to change many times during such a substitution, as intermediate quantities on the left of the sign $\rightarrow$ are computed and then used to obtain the value of the full expression. On the other hand, the state of the machine may not change at all, as in the case of a substitution $n + 1 \rightarrow n$ if the value of n itself does not have to be computed or stored: in such a case numerical quantities associated with compound variables like x_n have their names changed while they are sitting quietly in the storage of the computer. E.g. the number that was formerly denoted by x_n will have to be called x_{n-1} the next time it is referred to. We wish to emphasize again that the flow diagram is intended to give an abstract description of a calculation, not a coding procedure.

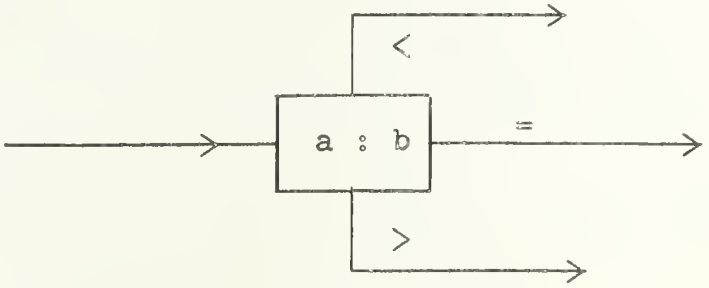
In contrast to substitution, by means of which a calculation progresses, we have <u>statements</u>, containing relation signs like $=$, $<$, $>$, $\leq$, $\geq$, $\equiv$, $\not\equiv$, etc. which are simply either true or false but do not indicate that any variable is to have its value changed. Statements are of two types -- <u>absolute</u> and <u>conditional</u>.

Rule 4. <u>An absolute statement has no function or effect other than to remind the reader of something he may have forgotten or to point out the physical or other interpretation of a particular part of a calculation</u>. (Therefore it need not be complete or precise.) <u>It should be put in parentheses</u> and preferably beside a box or a flow line rather than inside a box. For example, if an induction on $n = 1, 2, \ldots, N$ is terminated not by testing

whether $n = N$ but by some other test that is passed only when $n = N$, one may wish to write the statement "$(n = N)$" near the exit line of the subroutine, viz., $\xrightarrow{(n = N)}$, or one may wish to include a descriptive phrase like "(fission)" or "(inelastic scattering)", to tell the physical significance of a test or of the calculation in a particular box.

Rule 5. A conditional statement is a statement occuring in a box having two exits, labelled "yes" and "no" (or equivalently), it being understood that the calculation takes the first exit if and only if the statement is true. Some people put a question mark after a conditional statement, and some do not. A conditional statement may contain just one variable, as "$x > 0$" or several, as "$k \equiv \ell \pmod{n}$". For simplicity and economy of space, one often combines a condition box with the operation box that precedes it, as in the example on page 17. In this case the conditional statement must be the last item in the box.

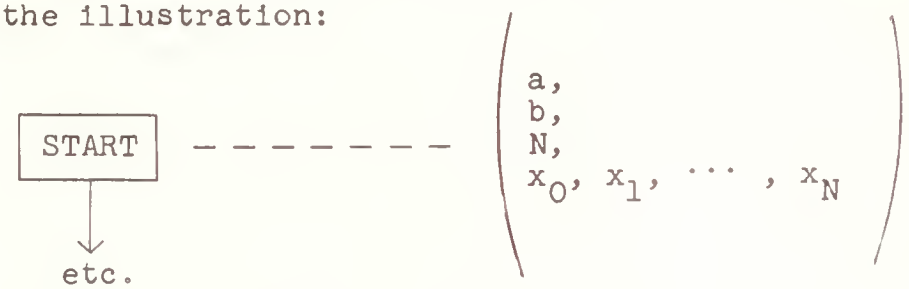
Multiple branching can be achieved by testing one condition after another. However, one case is sufficiently common to deserve a special notation; in it, ":" is used as a "compare" symbol, and the exit lines are labelled with relation symbols, as follows.



This device can be used whenever the conditional statements are mutually exclusive and each can be expressed by a relation symbol standing between two specified variables.

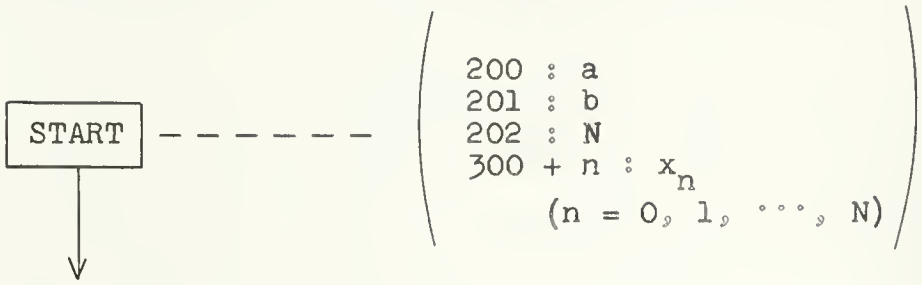
Occasionally, a computer is allowed to calculate for a very brief period with undefined quantities (e.g. with whatever numbers happen to be left over in the computer from previous operations) if one can be sure that the quantities so calculated will either be ignored or multiplied by zero. For example, the first step of an iteration may require only part of a subroutine used in later steps, but it may simplify the coding (at the expense of a slight increase in computing time) to allow the computer to go through the entire subroutine. Similar occurrences may be allowed also in an abstract flow diagram; but otherwise every symbol occurring in an expression on the left of a sign $\rightarrow$ should have had some numerical value assigned to it previously: i.e., it should denote either the result of some previous step or an input datum.

Rule 6. Variables denoting the input data of a problem should be written in a list, in a special kind of box resembling a huge pair of parentheses and connected to the flow line by a dashed line, as in the illustration:



The input data are the free variables of a calculation. If storage locations (either absolute or symbolic) for a particular

program are also indicated, the list becomes a storage chart, viz.

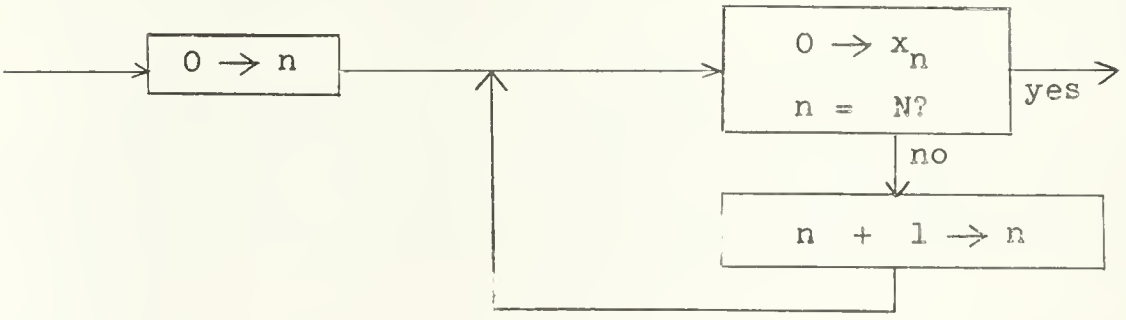


Similar lists or storage charts may appear elsewhere in the diagram.

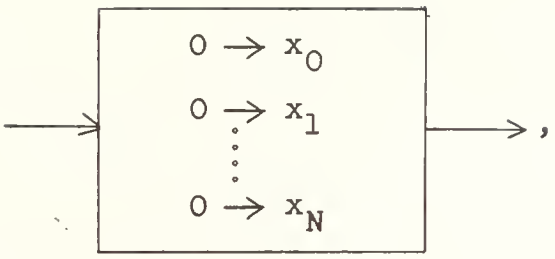
Further conventions may be used, more or less at will, provided the meaning is clear. For example, the expression

$$\text{"}0 \rightarrow x_n \quad (n = 0, 1, \cdots, N)\text{"} ,$$

occurring in a box, does not conform to the preceding rules, but is an obvious abbreviation for the steps:



However, great care must be exercised to notice exactly which variables have their values changed. The above set of steps is different from the set



and may actually lead to different results in a calculation, because the variable n generally has its value changed in the first case but does not even appear in the second. Similarly, the steps $x^2 \rightarrow y$, $y^2 \rightarrow z$ will generally have a different effect from the single step $x^4 \rightarrow z$ if the symbol y has had a value assigned to it previously and is to be made use of subsequently. It is for this reason that we regard a generalized substitution as the basic step of an abstract description of a calculation.

Two more subjects must be discussed before we are finished with flow diagrams: they are remote connections and a special symbolism, used in Monte Carlo work, to denote the act of sampling a random distribution.

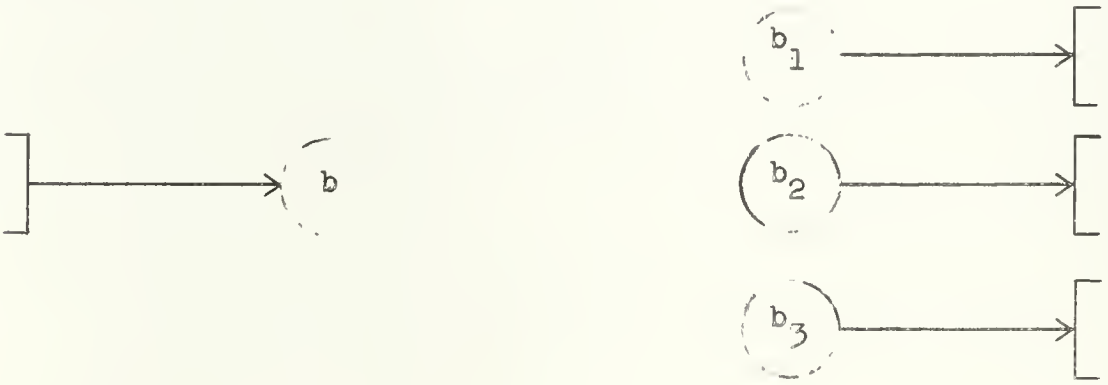
A remote connection is simply the replacement of a long flow line, viz.,

by the following diagramatic device,

which has the same meaning. The flow jumps from the first circle, called the exit point, to the second circle, called

the entrance point, or, more generally to that particular entrance point bearing the same symbol as the exit point that it came from (the letter "a" above).

This device is convenient if the flow diagram occupies two or more sheets of paper. Furthermore, it can be made into a variable remote connection, by having several possible entrances, e.g.,



so that the flow can jump to any one of the entrances, depending on circumstances (this is simply a convenient way of achieving multiple branching). The flow diagram itself must somehow specify which alternative is to be followed. This is done by regarding the symbol of the exit point as a variable whose value is to be established by a substitution according to the same rules as for numerical variables. Thus, the substitution

$$b_2 \rightarrow b \ ,$$

in a box of the flow diagram, means that henceforth, until further notice, the variable exit point b is to be identified with the entrance point b_2. Variable connections

are thus always prepared in advance and can be modified at any time. A few obvious rules are

Rule 7. No two entrance points may have the same symbol.

Rule 8. A variable exit may not have the same symbol as any entrance (for then it automatically establishes a fixed connection).

The general rules governing substitutions allow a great deal of flexibility in the use of variable connections. For example, if (a) and (b) are both variable exits and (b_1), (b_2), (b_3) are entrances, (a) can also be connected to any of the entrances by one of the substitutions

(b_1) → (a) ,

(b_2) → (a) ,

or (b_3) → (a) .

Furthermore, if, at some point in the flow diagram, we know that (b) has been connected to one of the entrances, we may wish to connect (a) to that same entrance without committing ourselves as to which of the entrances has been selected. This is done by a substitution

(b) → (a) ,

If the variable symbol (b) has previously had the value

(b_2) , say, assigned to it, the above substitution will then automatically establish a connection from (a) to (b_2) .

This completes the discussion of general flow diagrams and we come now to a special device used in Monte Carlo work. In Chapter IV, the generation of random variables was discussed, and it was noted that cursively written (script) letters are used to denote probability distributions; in particular, the symbol $\mathcal{U}$ denotes the uniform distribution of a single variable in the interval (0, 1). The expression

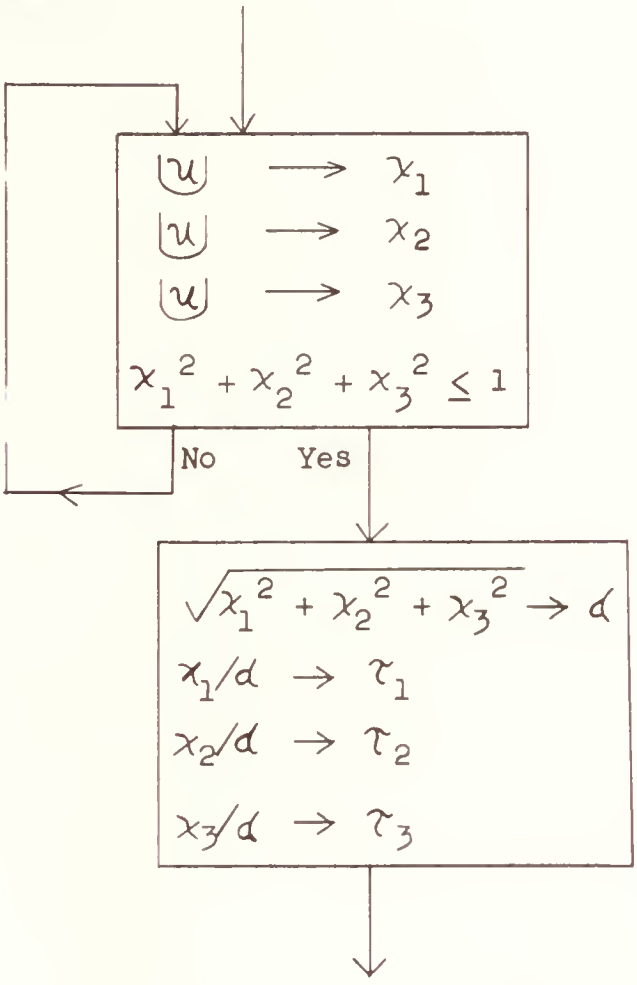
$$|\mathcal{U}| \rightarrow x$$

means that a number in the interval (0, 1) is drawn at random from $\mathcal{U}$ (and independently of all such numbers previously drawn from it), and that henceforth, until further notice, this number will be denoted by x. This symbolism represents the idealized concept, appropriate for an abstract description of a Monte Carlo calculation, of drawing x from a mathematically random and uniform distribution; in practice, it is approximated by setting x equal to the next number of a pseudo-random sequence, or in some similar fashion.

Similarly, if "$\mathcal{Sph}$" (for "spherical") denotes the distribution of unit vectors having random directions in three-dimensional space, the expression

$$|\mathcal{Sph}| \rightarrow \tau$$

means that a vector is drawn from this distribution and denoted[7] by $\vec{\tau}$. In practice, this might be accomplished by means of the steps



where τ_1, τ_2, τ_3 are the components of $\vec{\tau}$; but the former expression is more compact and allows for more efficient ways of generating the distribution "Spk", if they can be found.

When such conventions are used, the flow diagram describes an abstract stochastic process, or idealized Monte Carlo procedure,

---

[7] There is no objection to the use of vector notation in flow diagrams; the rules for calculating with vectors are perfectly definite.

rather than a deterministic calculation. In use, it must be "connected to" suitable sources of random variables, which in practice are not quite ideal, at least conceptually.

In the margin of a flow diagram (or on a separate sheet) one should identify all symbols denoting special probability distributions, all symbols denoting special functions (e.g. Bessel functions), etc. The flow diagram then gives a concise specification of an abstract calculational procedure. Just how well this procedure can be carried out on an actual computer may still depend on such mundane things as rounding errors and the size of the memory.

3. The Estimation of Significance

Until it has been ascertained whether rounding-errors have destroyed the usefulness of a calculation, the job is not finished. The task of a computer (electronic or human) consists of two parts: 1) performing the right operations on the right operands in the right order; 2) assessing the error introduced, at each stage, by the imperfections (limited number of decimal or binary places), of the arithmetical devices used. Most modern machines perform part 1) rapidly, efficiently, and automatically and do nothing whatever about part 2). In this section I shall discuss various methods of taking care of part 2), also automatically, during a calculation.

Note 1. I have in mind problems in which the basic mathematical context is that of the real number field. Work with integers can of course be done exactly.

Note 2. This discussion makes sense only in the framework of floating-point work, except for simple problems in which no scaling is required. As soon as numbers have to be shifted left and right to keep them in scale, automatic procedures become imperative, if one wishes to know what is really happening to significant digits.

Method 1. A calculation is done twice, with standard normalized floating point; in the second version, the treatment of the rightmost bit of each number is altered, either randomly or by some procedure different from the standard rounding. Comparison of the two sets of answers gives an indication of which digits are significant. This method is advocated by Bengt Carlson and will be used on "STRETCH". (See also Forsythe.[7])

Method 2. A number is represented by a set of three quantities: a fractional part M, an exponent e, and a significance index s. M is a binary or decimal fraction ($\frac{1}{2} \leq |M| < 1$ or $\frac{1}{10} \leq |M| < 1$, respectively - i.e., the number is normalized), s is an integer (≥ 0) equal to the number of digits of M that are significant (or alleged to be significant according to a certain set of rules). Together, M, s, and e represent the number $(M \pm 2^{-s})2^{e}$ or $(M \pm 10^{-s})10^{e}$.

The rules are: 1) whenever a number is being denormalized, in preparation for addition or subtraction, (this happens to the summand with the smaller exponent), its index is increased[8]

[8] but never outside the limits $(0, s_0)$, where s_0 is the maximum value of s allowed by the word-length of the machine.

by the same amount as the exponent, so that the indicated error $2^{-s}2^{e}$ or $10^{-s}10^{e}$ is unchanged. 2) then, in addition or subtraction, the index s of the result is set equal to the smaller of the indices of the summands (after step 1). 3) after addition or subtraction, during normalizing (in case of cancellation) or a one-place right shift (in case of overflow), the index is again changed[8] by the same amount as the exponent. 4) In multiplication and division, after the complete operation, including any final adjustment for overflow or normalizing, the index is set equal to that of the less accurate operand. 5) Division by a number with $s = 0$ is an error, regardless of the values of e and M, for then zero is included in the possible values of the denominator -- i.e. in the range $(M \pm 1)2^{e}$ or $(M \pm 1)10^{e}$.

This method has been used by the author[1] and is being wired into the "GEORGE" machine at the Argonne National Laboratory.[5]

Method 3. Standard floating point representation is used, except that each number is allowed to have just enough leading zeros after the binary or decimal point so that the remaining digits are all significant (with possible exception of a small fixed number of guard digits, at the extreme right, to absorb the accretion of rounding errors - of course, the machine need not know which digits are guard digits and which are truly significant ones.) Addition and subtraction are unnormalized; after multiplication and division the result is shifted, right or left, until it has exactly the same number of leading zeros as the less accurate operand, and its exponent is correspondingly adjusted.

This method has been used by Glenn Lewis,[2] and a very similar method is being wired into Maniac III at Chicago.

Method 4. This is the physicist's way of refining method 2 or 3; it is reported to have been proposed by L. H. Thomas. It is probably unnecessarily refined for most purposes and uncomfortably complicated for wiring into a machine, but would occasionally be of value if it is available via subroutines or microprogramming. A number is represented by three quantities M, δM, and e, where M and δM are both binary or decimal fractions; the number represented is $(M \pm \delta M)2^e$ or $(M \pm \delta M)10^e$, where δM denotes the probable error. When numbers are combined arithmetically, the errors of the operands are assumed to have normal probability distributions, and the standard formulas are used for the probable error of the result. To save bits, δM may itself be represented as $(\delta m)2^{e'}$ or $(\delta m)10^{e'}$, where δm is then kept with an accuracy of say 8-10 bits. To avoid root-extraction, a simple approximation is used for $\sqrt{y_1^2 + y_2^2}$.

Method 5. ("Range arithmetic") This is the mathematician's way of refining method 2 or 3. It has been proposed and tested, by use of subroutines, by (R?) Moore.[3] A number χ is represented by two normalized floating point quantities χ_1 and χ_2, which determine a narrow interval that contains the true value of χ; $\chi_1 \leq \chi \leq \chi_2$. (Normally, the exponents of χ_1 and χ_2 would be equal and it would suffice to store just one of them.) For each input datum, one sets $\chi_1 = \chi_2 = \chi$. When numbers are combined arithmetically, the lower limit and the upper limit of the result are calculated separately in such

a way that for each quantity in the calculation an interval is obtained in which its true value is guaranteed to lie.

Method 6. Certain specialized problems deal with rational numbers only. The number $\chi = p/q$ is represented by the integers p and q, which are stored. In addition to +, -, x, ÷, there is provision for reduction to lowest terms (by some variation of the euclidean algorithm). This method may possibly be used by Charles Lytle in the investigation of certain compatibility conditions connected with the theory of pseudo-analytic functions. I learned recently of an application of group theory to quantum mechanics, in which it was found useful to compute with a similar precise representation of certain algebraic numbers in a machine.

Comments. a. Methods 2 and 3 appear to be practically equivalent in performance. They have both been tested, at N.Y.U., on problems of varying complexity, using power series, where cancellation is a notorious worry. Judged by all available checks (including problems for which the exact solution is known), they give a quite accurate indication of the true significance; the number of actually correct digits is almost always the same, give or take one, as the number claimed to be significant. Similar tests have been reported recently, for method 2, by workers at the Argonne National Laboratory;[5] they report that the method is equally successful for problems of matrix inversion.

b. In a problem requiring the addition of a very large number N of quantities all of comparable accuracy and magnitude, methods 2 and 3 would underestimate the errors by failing to take

into account their accumulation, which tends to increase them by about $\sqrt{N}$. This does not appear to happen appreciably in power-series or matrix work, and I suspect that it only rarely happens otherwise, but it might happen, for example, in the numerical solution of an ordinary differential equation by very fine steps. Then, method 4 would be preferred.

c. For work in the physical sciences, where errors are estimated (in the physicist's sense) rather than given guaranteed bounds, method 4 would seem to be preferable to method 5 if an improvement over method 2 or 3 is really needed, because the guaranteed bounds may spread apart, during a calculation, considerably more rapidly than the most probable bounds.

d. Method 5 ought to be available for occasional rigorous tests of numerical methods. These might be sufficiently rare that use of a subroutine for the range arithmetic would suffice. However, it may turn out that it is precisely in very long calculations that one will want to make such tests; anyway, the method should be easily achieved in a micro-programming system.

e. The usefulness of method 6 (exact representation of rational numbers) is likely to be limited not only by the presence of irrational operations in many problems, but also by the tendency of the numerators and denominators to become progressively larger during a calculation.

References

[1] Richtmyer, R. D., Detached-Shock Calculations by Power Series, I, NYO-7973, October 15, 1957.

[2] Lewis, G., Two Methods Using Power Series for Solving Analytic Initial Value Problems, NYO-2881, January 1, 1960.

[3] Moore, (R?), by word of mouth. I was unable to find a reference to the work on range arithmetic.

[4] Carr, J. W. III, Error Analysis in Floating Point Arithmetic, Comm. Assoc. Computing Machinery, vol. 2, no. 5, p. 10, May, 1959.

[5] Gray, H. L. and Harrison, C. Jr., ANL, Normalized Floating-Point Arithmetic with an Index of Significance, 1959 Proceedings of the Eastern Joint Computer Conference.

[6] Ashenhurst, R. L., and Metropolis, N. C., Unnormalized Floating Point Arithmetic, Journal of the ACM, vol. 6, p. 415, July, 1959.

[7] Forsythe, G. E., Reprint of a Note on Rounding-off Errors, SIAM review, vol. 1, p. 66, 1959.

N. Y. U. Institute of
Mathematical Sciences
25 Waverly Place
New York 3, N. Y.